DRAWINGS:

3.1 Relational Database of the AC-knowledge:

3.1.1 ER-Diagram of the AC-Database:

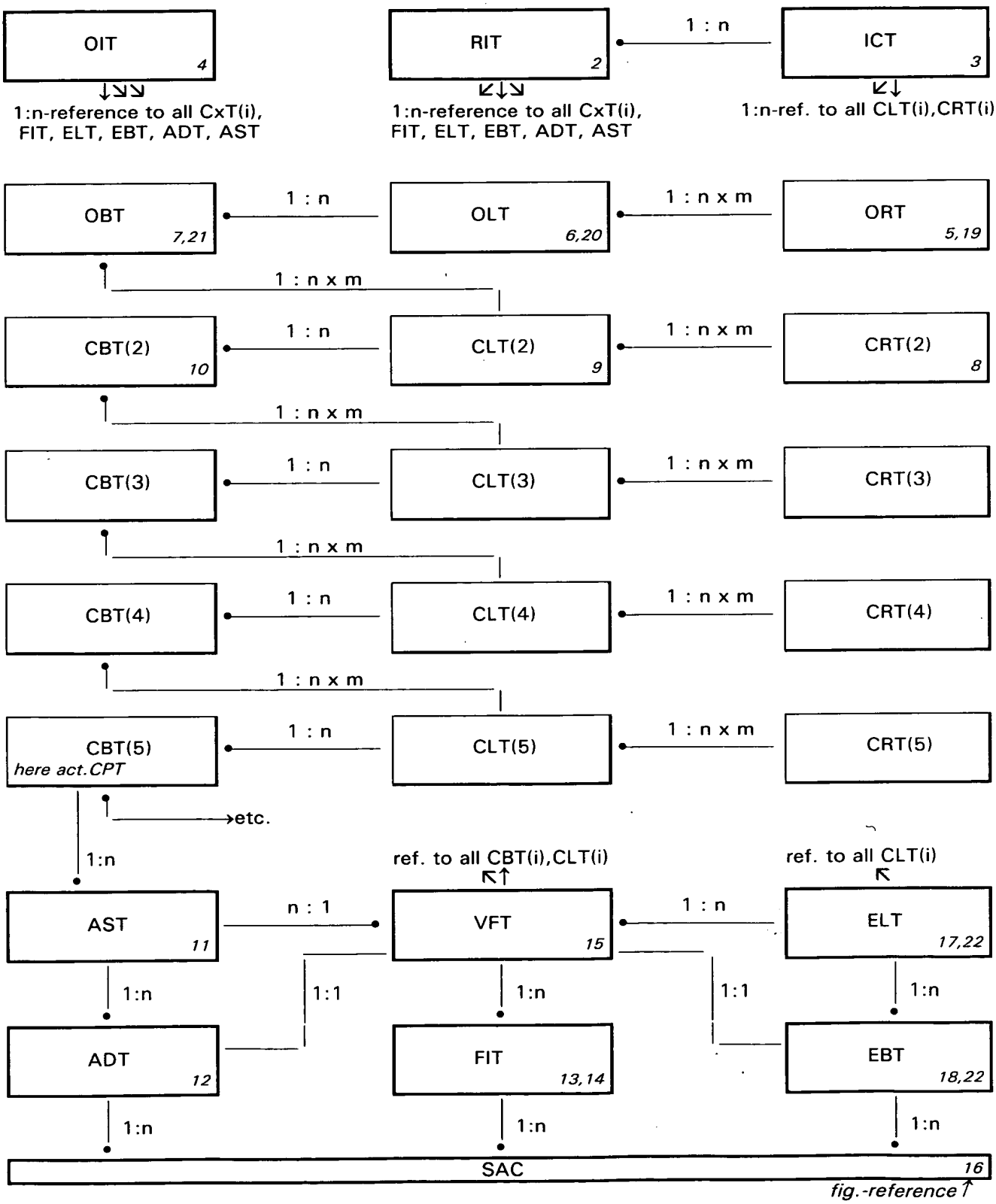

Fig. 1

3.1.2 Tables of the AC-Database:

Register-Identification-Table: *[RIT: every processor-register gets a Reg.ID and a Reg.BitCode]*

column:	*datatype*	*value-range*	*meaning:*
Register_ID *(PK)*	signed byte	-128..127	0 = Flags-reg., 1-... = data-reg., adr.reg., adr.reg.-destinations, FFP-reg., control-reg., debug-reg., etc.; neg.reg.ID = exception-vector-nr. (no processor-reg.)
Register_BitCode	number	$0..2^{128}-1$	2^Register_ID, 0 if Register_ID is negative.
Register_type	char(1)	1 Byte	*type of register:* # = flags-reg., D = data-reg., A = address-reg., V = adr.-reg.-destination, E = exception-vector, etc. (processor dependant).
Register_number	byte	0..127	current number of the *{register-type}*-registers.
Register_Description	varchar2(32	≤32 Bytes	optional description of the register[-reference].

Fig.2a

Register_ID	*Register_BitCode*	*Register_type*	*Register_number*	*Register_Description*
...	0	E	...	*{for all Exception-Vectors}*
-8	0	E	8	Privilege-Violation Exception
...	0	E	...	*{for all Exception-Vectors}*
0	1	#	0	Status-Register ($\triangleq$ $EFlags_{\pi}$)
1	2	D	0	Data-Register D0
...	4	D	...	*{for all Data-Registers D1-D6}*
8	8	D	7	Data-Register D7
9	16	A	0	Address-Register A0
...	...	A	...	*{for all Address-Registers A1-A6}*
16	65536	A	7	Address-Register A7 (= USP!)
17	131072	V	0	Destination of Address-Register A0
...	...	V	...	*{for all Adr.-Reg.-Destinations A1-A6}*
24	16777216	V	7	Destination of Adr.-Reg. A7 [= (USP)]
25	33554432	v	0	Destination before Adr.Reg A0 [-(A0)]
	...	v	...	*{for all Adr.Reg.-Dest. before A1-A6}*
32	$1.0000.0000	v	7	Destination before Adr.Reg A7 [-(USP)]
33	$2.0000.0000	F	0	Floating-Point Data-Register FPR0
...	...	...	...	*{for all further Registers}*

Fig.2b *(RIT in a Motorola-Example)*

Initial-Conditions-Table: *[ICT: every register(-reference) gets 62 initial conditions]*

column:	*datatype*	*value-range*	*meaning:*
IniConNr *(PK)*	signed byte	-31..+30	number of the initial-condition combination
Register_ID *(PK)*	signed byte	0..127	see RIT
Register_Value	integer	$0..2^{32}-1$	value of the register(-reference) on actual IniConNr

Fig.3a

Therefore 62*#registers test-values are generated, p.e. using the following function:
Register_Value(IniConNr, Register_ID) = SGN(IniConNr) * INT(2^ABS(IniConNr/2) + ½) + prime_number(3*Register_ID) or similar.
,where prime_number(0) = 0 and prime_number(-n) = -prime_number(n)
[no 2 equal register(-destination)-values]

Fig.3b

Operation-Identification-Table: *[OIT: every processor-operation gets an Oper.ID and an Oper.BitCode]*

column:	*datatype*	*value-range*	*meaning:*
Operation_ID *(PK)*	signed byte	-1..63	bit of the Calculation_BitCode - see table below.
Operation_BitCode *(FK)*	number	$0..2^{128}-1$	2^Calculation_ID - see table below.
Operation_Type	char(5)	5 Bytes	5 characters-code of the operation-type, see *Fig.4c*
Operation_Mnemonic	char(5)	5 Bytes	abbreviation of the operation - see table below.
Operation_Description	varchar2(32)	≤32 Bytes	optional description of the operation - see table below

Fig.4a

Operation_ID	*Operation_BitCode*	*Operation_Type*	*Op.Mnemonic*	*Operation_Description*
-1	0	.....	???	unknown operation
0	1	.I11?	TST	set flags in dependence of reg.(-ref.)
1	2	.I12!	NEG	negation|amount
2	4	.I12!	NOT	bitwise inversion
3	8	:I02_	MOVI	const.integer→register(-reference)
4	16	:I12+	ADDI	add constant integer
5	32	:I12-	SUBI	subtract constant integer
6	64	:I13*	MULI	multiply constant integer
7	128	:I23/	DIVI	divide by constant integer
8	256	:I13%	MODI	rest of integer-division
9	512	:I12*	SHLI	integer-times a duplication
10	1.024	:I12/	SHRI	integer-times a halvation
11	2.048	:I12|	ORI	set bits set in a constant integer
12	4.096	:I12&	ANDI	clear bits not set in a const. integer
13	8.192	:I12?	BTSTI	check if int-th bit is set in reg.(-ref.)
14	16.384	:I12?	CMPI	reg.(-ref.)-comparison with integer
15	32.768	II22_	MOV	move src.-reg.(ref.)→dest.reg(ref.)
16	65.536	II22+	ADD	addition of register(-reference)
17	131.072	II22-	SUB	subtraction of register(-reference)
18	262.144	II23*	MUL	multiplication of register(-reference)
19	524.288	II33/	DIV	division by register(-reference)
20	1.048.576	II33%	MOD	rest of division by register(-ref.)
21	2.097.152	II22*	SHL	register(-ref.)-times a duplication
22	4.194.304	II22/	SHR	register(-ref.)-times a halvation
23	8.388.608	II22|	OR	set bits set in of register(-reference)
24	16.777.216	II22&	AND	clear bits not set in register(-ref.)
25	33.554.432	II21?	BTST	check if reg.(-ref.)-th bit is set
26	67.108.864	II21?	CMP	compare reg.(-ref.)1 with reg.(-ref.)2
27	134.217.728	:P00.	JMP	add integer to PC_μ/EIP_π (=jump to)
28	268.435.456	CP1.<	JLT	jump if CMP <
29	536.870.912	CP1!>	JLE	jump if CMP ≤
30	1.073.741.824	CP1.=	JEQ	jump if CMP =
31	2.147.483.648	CP1!<	JGE	jump if CMP ≥
32	4.294.967.296	CP1!=	JNE	jump if CMP ≠
33	$2.0000.0000	CP1.>	JGT	jump if CMP >
34	$4.0000.0000	CP1!<	JPL	jump if ≥ 0
35	$8.0000.0000	CP1.<	JMI	jump if < 0
36	$10.0000.0000	CP1.^	JCS	jump if carry-flag is set
37	$20.0000.0000	CP1!^	JCC	jump if carry-flag is clear
38	$40.0000.0000	CP1.~	JVS	jump if overflow is set
39	$80.0000.0000	CP1!~	JVC	jump if overflow is clear
40	$100.0000.0000	CP2.<	DJMP	decrement and jump if reg.(-ref.) < 0
41	$200.0000.0000	PS1..	CALL	$PC_\mu/EIP_\pi \rightarrow -(USP_\mu/ESP_\pi)$; +JUMP
42	$400.0000.0000	SP11.	RET	$(USP_\mu/ESP_\pi)+ \rightarrow PC_\mu/EIP_\pi$
43	$800.0000.0000	.I...	I???	unknown integer-operation
44	$1000.0000.0000	.F...	F???	unknown floating-point-operation

45	$2000.0000.0000	FF09_	FINIT	initialise floating-point-unit
46	$4000.0000.0000	FI12_	FIST	store float.point-reg.→integer-reg.
47	$8000.0000.0000	IF12_	FILD	load integer-reg.→floating-point-reg.
48	$1.0000*$2^{32}$	IF22+	FIADD	floating-point add constant integer
49	$2.0000*$2^{32}$	IF22-	FISUB	floating-point subtract const.integer
50	$4.0000*$2^{32}$	IF22*	FIMUL	floating-point multiply const.integer
51	$8.0000*$2^{32}$	IF22/	FIDIV	floating-point divide by const.integer
52	$10.0000*$2^{32}$	IF21?	FICMP	float.pt.compare with integer→flags
53	$20.0000*$2^{32}$	:F02_	FLD#	load constant to floating-point-reg.
54	$40.0000*$2^{32}$	.F12!	FABS	build floating-point amount
55	$80.0000*$2^{32}$	FF12_	FLD	copy floating-point-register
56	$100.0000*$2^{32}$	FF22+	FADD	add 2 floating-point-register
57	$200.0000*$2^{32}$	FF22-	FSUB	subtract 2 floating-point-register
58	$400.0000*$2^{32}$	FF22*	FMUL	multiply 2 floating-point-register
59	$800.0000*$2^{32}$	FF22/	FDIV	divide one float.pt.reg. by another
60	$1000.0000*$2^{32}$	.F12@	FSQRT	square-root of a floating-point-reg.
61	$2000.0000*$2^{32}$	.F12@	FSIN	floating-point-sine
62	$4000.0000*$2^{32}$	.F12@	FCOS	floating-point-cosine
63	$8000.0000*$2^{32}$	.F12@	FATAN	floating-point arc-tangent
64	$1*$2^{48}$	FF22*	FEXP2	$y:=y*2^x$ (anykind of exp.-func.)
65	$2*$2^{48}$	FF22/	FLOG2	$y:=x*\log_2 y$ (any kind of logarithm)
66	$4*$2^{48}$	FF21?	FCMP	compare 2 float.-point-reg.→Flags
67	$8*$2^{48}$	$I11_	SMOV	move from a special Register
68	$10*$2^{48}$	I$11_	MOVS	move to a special Register
...	...	...	...	...

Fig.4b

... using the following operation-type character-codes:

1st char. = source , 2nd char. = dest.:	?	= unknown, ambitious letter for all possible following
	.	= nothing
	:	= constant number
	I	= integer-register-[reference]-value
	F	= floating-point register
	C	= condition-code register (last byte in $EFlags_\pi$/SR_μ)
	P	= instruction-pointer / program-counter (EIP_π/PC_μ)
	S	= stack-pointer reference-value
	~	= comparison-operation →flags
	$	= a special-register like flags-, control-, debug-, ... -reg.
	!	= logical NOT for comparison in the 4th field
3rd char. = number of source-registers:		including destination reg., if it's used as source-reg. too
4th char. = number of dest.-registers:		including flags-register but without the instruction-pointer
5th char. = effect of arithmetic:	?	= unknown
	.	= none
	!	= amount|negation|bitwise_inversion
	+	= addition
	−	= subtraktion
	*	= multiplication
	/	= division
	%	= rest of division
	|	= setting of bits (mostly bitwise OR)
	&	= clearing of bits (mostly bitwise AND)
	@	= trigonometric- or exponential-function
	>	= greater? (CC-flags dependant action)
	<	= less? (CC-flags dependant action)
	=	= equal? (CC-flags dependant action)
	^	= carry? (CC-flags dependant action)
	~	= overflow? (CC-flags dependant action)

Fig.4c

OpCode-Register-Table: *[ORT - by opcode icl, initial-conditions concerned registers and the effect]*

column:	*datatype*	*value-range*	*meaning:*
OpCode *(PK)*	integer	$0..2^{32}-1$	complete instruction, truncated if > 4 bytes
IniConNr *(PK)*	signed byte	-31..30	current number of the used initial conditions
Register_ID_dest *(PK)*	signed byte	0..127	one by execution concerned destination-reg. (see RIT)
Register_ID_source *(PK)*	signed byte	-1,0..127	-1 or one possible source-register (see RIT).
value_before_change	integer	$0..2^{32}-1$	register(-reference)-value before it was changed
value_after_change	integer	$0..2^{32}-1$	register(-reference)-value after changing
gradient_if_unsigned	signed byte	-128..127	before/after-gradient, when defined as unsigned
gradient_if_signed	signed byte	-128..127	before/after-gradient, when defined as signed
value_source	integer	$0..2^{32}-1$	value of a possible source-register(-reference)
Operations_BitCode	number	$0..2^{128}-1$	bitmask, which flags all possible operations between this Register_ID_dest / Register_ID_source -combination (p.e. 2+2=2*2 using same reg.'s). Values see CIT, calculation see fig.19.

Fig.5

For every register- or register-reference-modification of the same opcode-execution one entry is generated, which gives information about the register(-reference)-values before and after the execution and further information about the degree of changing and with it a hint to a possible operation and a possible source-register which were used. (Packed-, nibble-, or BCD-operations are not considered.)
The last address-register is the stack-pointer. The last data-register is the "energy"-register.
An address-register can be every register which value can be a pointer to memory which destination can be accessed using this register as a reference.

Several registers can be modified simultaneous - therefore this additional 1:n table, where *Register_ID_dest* means the identity of the changed register. Sometimes many register(-references) could be the source for one operation - this quantum increases through the sum over all possible operations.
Therefore the following tables identify the used opcode and the concerned register(s):

OpCode-Learn-Table: *[OLT - ascertained effect of the opcode using the concerning initial conditions]*

column:	*datatype*	*value-range*	*meaning:*
OpCode *(PK)*	integer	$0..2^{32}-1$	complete instruction, truncated if > 4 bytes
IniConNr *(PK)*	signed byte	-31..30	number of used initial condition
active_ChkSum_corrupt	boolean	1|0	flag: checksum of active AC-program changed
inactive_ChkSum_corrupt	boolean	1|0	flag: checksum of inactive AC-program changed
Exception_Vect_changed	signed byte	-128...0	Register_ID of the (first) overwritten exception-vector
multiple_Exc_Vect_chg	boolean	1|0	more than one exception-vector was overwritten
Processor_Mode_Changed	boolean	1|0	flag: processor-mode changed (p.e. trace cleared)
Number_of_Exception	byte	0..N+1	exception-number [0:=no_exception] (if 0=exc.:+1)
OpCode_length_or_jump	signed byte	-128..127	EIP_π/PC_μ after execution − EIP_π/PC_μ before execution -128=$FF=long back-jump, 127=$7F=long forward j.
CCR_before_execution	byte	0..255	CC-flags, which could cause a jump.
Register_changed_BitCode	number	$0..2^{128}-1$	Ξ 2^ORT.Register_ID_dest ∀ ORT(opcode,IniConNr)
Register_source_BitCode	number	$0..2^{128}-1$	Ξ 2^ORT.Register_ID_source ∀ ORT(opcode,IniConNr)
max_Operations_BitCode	number(19)	$0..2^{128}-1$	Ξ ORT.Calculation_BitCode ∀ORT(opcode,IniConNr)
min_Operations_BitCode	number(19)	$0..2^{128}-1$	ζ ORT.Calculation_BitCode ∀ORT(opcode,IniConNr)
time_of_execution	integer	$0..2^{32}-1$	deci-seconds_after_{20.9.1994, 0:00:00,0 o'clock}
cycles_of_execution	byte	1..255	clock-cycles the opcode-execution needed
aim_valuation	signed byte	-128..127	aim-attaining-valuation using above initial-conditions
gradient_aim_valuation	signed byte	-128..127	-"-difference to *CLT(n-1, IniConNr).aim_valuation*

Fig.6

OpCode-Base-Table: *[OBT - ascertains the effect of the opcode-execution from the initial conditions]*

column:	*datatype*	*value-range*	*meaning:*
OpCode *(PK)*	integer	$0..2^{32}-1$	complete instruction, truncated if > 4 bytes
Execution_counter	byte	0..255	number of the OLT-entries until now
FatalError_counter	byte	0..255	number of the opcode-caused fatal errors until now: CheckSum_corrupt, Exception_Vect_changed, Trace_Bit_cheared, Processor_Mode_changed, and the exceptions without *Divide-Error, Overflow.*
low_Error_counter	byte	0..255	number of *divide*-errors plus *overflow*-exceptions
Jump_longOp_probability	signed byte	-128..127	probability that it's a long opcode or a jump
avg_OpCpde_jump_length	signed byte	-128..127	average length of opcode or jump
OpCode_len_unconfirmed	boolean	1|0	min. one divergence from above average exists
avg_cycles_of_execution	byte	1..255	average by opcode-execution needed clock-cycles
exec_cycles_unconfirmed	boolean	1..0	min. one divergence from above average exists
Register_write_probability	signed byte	-128..127	probability: opcode writes into a register
Register_copy_probability	signed byte	-128..127	probability: opcode copies register
Memory_write_probability	signed byte	-128..127	probability: opcode writes into memory
Memory_copy_probability	signed byte	-128..127	probability: opcode copies memory
Reg_to_Mem_probability	signed byte	-128..127	probability: opcode copies reg. to adr.reg.-destination
Mem_to_Reg_probability	signed byte	-128..127	probability: opcode copies adr.reg.-destination to reg.
Multi_Reg_write_prob	signed byte	-128..127	probability: opcode writes into more than one register
Multi_Mem_write_prob	signed byte	-128..127	probability: opcode writes to many adr.reg.-destination
Multi_Reg_to_Mem_prob	signed byte	-128..127	prob.: opcode copies ≥2 reg. to ≥2 adr.reg.-destination
Multi_Mem_to_Reg_prob	signed byte	-128..127	prob.: opcode copies ≥2 adr.reg.-destinations to ≥2 reg
all_Reg_dest_BitCode	number	$0..2^{128}-1$	ΞOLT.Register_changed_Bitcode ∀ OLT(OpCode)
cut_Reg_dest_BitCode	number	$0..2^{128}-1$	ζOLT.Register_changed_Bitcode ∀ OLT(OpCode)
all_Reg_source_BitCode	number	$0..2^{128}-1$	ΞOLT.Register_source_Bitcode ∀ OLT(OpCode)
cut_Reg_source_BitCode	number	$0..2^{128}-1$	ζOLT.Register_source_Bitcode ∀ OLT(OpCode)
max_Operation_BitCode	number	$0..2^{128}-1$	ΞOLT.max_Operation_BitCode ∀ OLT(OpCode)
min_Operation_BitCode	number	$0..2^{128}-1$	ζOLT.min_Operation_BitCode ∀ OLT(OpCode)
all_Operation_BitCode	number	$0..2^{128}-1$	ΞOLT.min_Operation_BitCode ∀ OLT(OpCode)
cut_Operation_BitCode	number	$0..2^{128}-1$	ζOLT.max_Operation_BitCode ∀ OLT(OpCode)
max_write_value	integer	$0..2^{32}-1$	maximum of all destination-values
min_write_value	integer	$0..2^{32}-1$	minimum of all destination-values
avg_write_value	integer	$0..2^{32}-1$	average over all destination-values
max_write_gradient	integer	$0..2^{32}-1$	maximum gradient of the changed value
min_write_gradient	integer	$0..2^{32}-1$	minimum gradient of the changed value
avg_write_gradient	integer	$0..2^{32}-1$	average gradient of the changes value
evaluated_source_Register	signed byte	-1,0..127	ascertained source-register-ID (after OBT-evaluation)
evaluated_source_NumReg	signed byte	-128, -1, 0..127	-128 ≙ LOB means source-constant; 0..127 means a further source-register ID; none=-1 (after OBT-eval.)
evaluated_dest_Register	signed byte	-1, 0..127	ascertained destination-register after OBT-evaluation
evaluated_dest_Register2	signed byte	-1, 0..127	possible 2nd dest.-reg. after OBT-evaluation or flags-reg. (2 real dest.-reg. ⇒ flags-reg. not appreciated).
evaluated_Operation_ID	signed byte	-1,0..63	ascertained operation-ID (after OBT-evaluation)
Confirmation_counter	byte	0..255	counter: same effects on other initial conditions
max_aim_valuation	signed byte	-128..127	max. valuability of the opcode for aim-attaining
avg_aim_valuation	signed byte	-128..127	average valuability of the opcode for aim-attaining
max_grad_aim_valuation	signed byte	-128..127	max. gradient of aim-attaining in relation to the by 1 shorter opcode-combination of the last CBT(i-1).
avg_grad_aim_valuation	signed byte	-128..127	average gradient concerning above -"-

Fig.7

Datatypes: *Boolean* 1 Bit, BCD/Nibble 4 Bit, *Byte/char(1)* 8 Bit, *Word/short* 16 Bit, *DWord/Integer* 32 Bit, *QWord/number(19)* 64 Bit, *number/number(38,0)* 128 Bit (38 digits ≙ 16 bytes), *varchar2(N)* string of variable length with max.N characters, *long* very long string with max(longDef) characters.

The following **combination-tables** are created dynamically - they have the same non-PK-columns, like the OBT respectively OLT respectively ORT, but for every additional number of combinations a further more opcode in the PK:

Combination-Register-Table: *[CRT(i), i = number of opcodes in the combination, CRT(1) = ORT]*

column:		*datatype*	*value-range*	*meaning:*
OpCode_1	*(PK)*	integer	$0\text{-}2^{32}\text{-}1$	opcode #1 (first of the combination)
{for all OpCodes}	*(PK)*	*all* integer	$0\text{-}2^{32}\text{-}1$	{for all opcodes from #2 up to N-1}
OpCode_N	*(PK)*	integer	$0\text{-}2^{32}\text{-}1$	opcode#N (last of the combination)
IniConNr	*(PK)*	signed byte	-31..30	current number of the used initial conditions
Register_ID_dest	*(PK)*	signed byte	0..127	one by execution concerned destination-reg. (see RIT)
Register_ID_source	*(PK)*	signed byte	-1..127	-1 or one possible source-register (see RIT).
{same non-PK columns like in the Opcode-Register-Table.}		see above	see above	*same non-PK columns like ORT.*

Fig.8

Combinations-Learn-Table: *[CLT(i), i = number of opcodes in the combination, CLT(1) = OLT]*

column:		*datatype*	*value-range*	*meaning:*
OpCode_1	*(PK)*	integer	$0\text{-}2^{32}\text{-}1$	opcode #1 (first of the combination)
{for all OpCodes}	*(PK)*	*all* integer	$0\text{-}2^{32}\text{-}1$	{for all opcodes from #2 up to N-1}
OpCode_N	*(PK)*	integer	$0\text{-}2^{32}\text{-}1$	opcode#N (last of the combination)
IniConNr	*(PK)*	signed byte	-31..30	current number of the used initial conditions
{same non-PK columns like in the Opcode-Learn-Table.}		see above	see above	*same non-PK columns like OLT.*

Fig.9

Combinations-Base-Table: *[CBT(i), i = number of opcodes in the combination, CBT(1) = OBT]*

column:		*datatype*	*value-range*	*meaning:*
OpCode_1	*(PK)*	integer	$0\text{-}2^{32}\text{-}1$	opcode #1 (first of the combination)
{for all OpCodes}	*(PK)*	*all* integer	$0\text{-}2^{32}\text{-}1$	{for all opcodes from #2 up to N-1}
OpCode_N	*(PK)*	integer	$0\text{-}2^{32}\text{-}1$	opcode#N (last of the combination)
{same non-PK columns like in the Opcode-Base-Table.}		see above	see above	*same non-PK columns like OBT.*

Fig.10

CBT(max.) = CPT = Combination-Plan-Table = point of origin of the outcoming program.

Programming-aim and valuation-function tables:

Aim-Solution-Table: *[AST - solutions of all programming-aims]*

column:	*datatype*	*value-range*	*meaning:*
Aim_ID *(PK)*	short	0..65535	Identifier of the programming-aim
Solution_Nr *(PK)*	byte	0..255	number of the solution-program
aim_Program	long	String	opcode-combination of the solution here as a string
Program_length	short	1..65535	length of the solution-program in doublewords
cycles_of_execution	integer	$1..2^{32}-1$	execution-time in clock-cycles of the solution-program
used_Registers_BitCode	number	$1..2^{128}-1$	bitcode of all in the solution-program used registers
used_Operations_Bitcode	number	$1..2^{128}-1$	bitcode of all in the solution-program used opcodes
used_aim_Valuation_Func	signed short	0..32767	identifier of the used aim-distance valuation-function

Fig. 11

Aim-Description-Table: *[ADT - identification and description of programming-aim]*

column:	*datatype*	*value-range*	*meaning:*
Aim_ID *(PK)*	short	0..65535	Identifier of the programming-aim
aim_Description	varchar2(32)	≤32 Bytes	description of the programming-aim
used_Processor_Mode	integer	$0-2^{32}-1$	flags above CC | control-register-bits
all_dest_Register_BitCode	number	$1..2^{128}-1$	bitcode of all output-registers in this task
all_source_Register_BitCode	number	$1..2^{128}-1$	bitcode of all input-registers in this task
unused_Regiser_BitCode	number	$1..2^{128}-1$	bitcode of all registers which should not be used
unused_Operation_BitCode	number	$0..2^{128}-1$	bitcode of all opcode-IDs which are not allowed to use in this task (default = $0000.0000:0000.0000)
aim_implement_solutions	long	String	string of the Aim_ID's (words) of earlier solutions, which could be implemented here.
aim_fulfill_valuation_mode	boolean	0|1	mode of aim-valuation: 0 = SQL ; 1 = machine-code
aim_fulfilled_Flag_Function	varchar2(99)	≤99 Bytes	boolean aim-attained recognition-function as a string
aim_Valuation_FunctionID	signed short	0..32767	identifier of the valuation-function (see VFT)

Fig. 12

Functions-Identification-Table: *[FIT - table of the basic subfunctions used in the valuation-function]*

a.) for SQL-functions:

column:	*datatype*	*value-range*	*meaning:*
Function_ID *(PK)*	signed byte	-1..127	identification-number of the basic function
Function_BitCode	number(19)	$0..2^{64}-1$	bitcode of this basic function (only one bit is set)
Function_Name	char(5)	5 Bytes	function-name
Function_Type	byte	0..99	0 = value, 1 = unitary, 2 = binary, 3 = ternary, ...
Function_Flatten	signed byte	-127..127	degree of flattening [+ = steepening, – = flattening]
Function_Template	varchar2(99)	≤99 Bytes	SQL function-template
Function_Description	varchar2(99)	≤99 Bytes	optional description of the basic sub-function

Fig. 13a

F.ID	*Function_BitCode*	*F.Name*	*F.T.*	*F.F.*	*Function_Template*	*Function_Description*
0	1	NUM	0	0	< following_value >	a constant number follows
1	2	ENGY	0	0	ELT.*energy_after*	energy after execution
2	4	GRAD	0	0	ELT.*energy_after* −ELT.*energy_before*	energy-gradient
3	8	VALUE	0	0	CLT(n). < columnNr >	value in the following column-number (last row)
4	16	EREG	0	0	< EnergyRegister_ID >	ID of the energy-register
5	32	SGN	1	0	SIGN(%s)	algebraic sign
6	64	ROUND	1	0	ROUND(%s, 0)	rounded
7	128	INT	1	0	FLOOR(%s)	truncated after dec.point
8	256	ABS	1	0	ABS(%s)	amount
9	512	NEG	1	0	−(%s)	negation
10	1.024	ADD	2	1	((%s) + (%s))	addition
11	2.048	SUB	2	-1	((%s) − (%s))	subtraction
12	4.096	MUL	2	4	((%s) * (%s))	multiplication
13	8.192	DIV	2	-4	((%s) / (%s))	division
14	16.384	MOD	2	-2	MOD(%s, %s)	rest of division
15	32.767	SQRT	1	-8	SQRT(%s)	square-root
16	$1.0000	CBRT	1	-12	POWER(%s, 1/3)	cube-root
17	$2.0000	MIN	2	-10	LEAST(%s, %s)	minimum
18	$4.0000	MAX	2	-10	GREATEST(%s, %s)	maximum
19	$8.0000	LN	1	-48	LN(%s)	natural logarithm
20	$10.0000	EXP	1	48	EXP(%s)	nat. exponential-function
21	$20.0000	LD	1	-32	LOG(2, %s)	logarithm on base 2
22	$40.0000	POT2	1	32	POWER(2, %s)	2nd power of ...
23	$80.0000	SIN	1	-64	SIN(%s)	sine
24	$100.0000	COS	1	-64	COS(%s)	cosine
25	$200.0000	TAN	1	127	TAN(%s)	tangent
26	$400.0000	ASIN	1	127	ASIN(%s)	arc sine
27	$800.0000	ACOS	1	127	ACOS(%s)	arc cosine
28	$1000.0000	ATAN	1	-127	ATAN(%s)	arc tangent
29	$2000.0000	SINH	1	40	SINH(%s)	sine hyperbolic
30	$4000.0000	COSH	1	50	COSH(%s)	cosine hyperbolic
31	$8000.0000	TANH	1	-127	TANH(%s)	tangent hyperbolic
32	$1.0000.0000	LOG	2	-64	LOG(%s, %s)	logarithm
33	$2.0000.0000	POT	2	64	POWER(%s, %s)	n-th power of ...
34	$4.0000.0000	OR	2	1	((%s) | (%s))	bitwise OR
35	$8.0000.0000	AND	2	-1	((%s) & (%s))	bitwise AND
36	$10.0000.0000	EQ	2	-127	DECODE(%s, %s, 1, 0)	equal
37	$20.0000.0000	LE	2	-127	DECODE(GREATEST(%s − %s, 0), 0, 1, 0)	less-equal
38	$40.0000.0000	GE	2	-127	DECODE(LEAST(%s −%s, 0), 0, 1, 0)	greater-equal
39	$80.0000.0000	FRAME	1	-10	GREATEST(LEAST(%s, +127), -128)	frame to signed-byte: max. = 127, min. = -128
40	$100.0000.0000	BITS	1	-64	(1 & %s) + (2 & %s)/2 + (4 & %s)/4 + (8 & %s)/8 + ...	number of bits in the integer value
41	$200.0000.0000	S_REG	0	0	ADT.*all_source_Registers-BitCode*	bitcode of the source-register
42	$400.0000.0000	D_REG	0	0	ADT.*all_dest_Registers_BitCode*	bitcode of dest.-register
43	$800.0000.0000	AIM_F	0	0	VAL(ADT.*aim_fulfilled_Flag-Function*)	result of the boolean aim-fulfilled-function
...	...	...	..	...	...	...

Fig. 13b

b.) for machine-code functions:

column:	*datatype*	*value-range*	*meaning:*
Function_ID *(PK)*	signed byte	-1..127	identification-number of the basic function
Function_BitCode	number(19)	$0..2^{64}-1$	bitcode of this sub-function
Operations_BitCode	number	$0..2^{128}-1$	bitcode of the used opcodes in this function
Registers_BitCode	number	$0..2^{128}-1$	bitcode of the used registers in this function
Function_Name	char(5)	5 Bytes	short notation of this sub-function
Function_Type	byte	0..99	0 = value, 1 = unitary, 2 = binary, 3 = ternary, ...
Function_Flatten	signed byte	-128..127	degree of function-flattening ($1 \triangleq f(x) = x$)
Function_OpCodes	number	$1..2^{128}-1$	sub-function in machine-code
Function_Description	varchar2(99	≤99 Bytes	optional description of the sub-function

Fig. 14a

Func.ID	*Func.BitCode*	*Oper.BitCode*	*Reg.BitCode*	*Func.Name*	*F.T.*	*Func.OpCodes*	*Function_Descript.*
0	1	$A000.4008	< energy >	FRAME	1	s.b. Func.1	prevent overflow
1	2	$28800.0009	< energy >	SGN	1	s.b. Func.2	signum
2	4	$0000.0002	< energy >	NEG	1	<NEG>	negation
3	8	$0000.0200	< energy >	MUL2	1	<SHLI>	division by 2
4	16	$0000.0400	< energy >	DIV2	1	<SHRI>	multiplication by 2
5	32	$0000.0100: 4A00.8018	⟨D0⟩|⟨en⟩	ILOG2	1	s.b. Func.3	mogarithm dualis
6	64	$1000.C000: 0000.0000	⟨FP0⟩|⟨en⟩	ISQRT	1	s.b. Func.4	square-root
7	128		s. 1.4.2	ICBRT	1	s.above 1.4.2	cube-root
8	256	$0000.8000	⟨en-1⟩|⟨en⟩	MOV	2	<MOV>	copying of one reg. before energy-reg.
9	512	$0000.8000	⟨en-1⟩|⟨en⟩	SWAP	2	s.b. Func.5	swap with reg. before energy-reg.
10	1024	$0001.0000	⟨en-1⟩|⟨en⟩	ADD	2	<ADD>	addition with -"-
11	2048	$0002.0000	⟨en-1⟩|⟨en⟩	SUB	2	<SUB>	subtraction of -"-
12	4096	$0004.0000	⟨en-1⟩|⟨en⟩	MUL	2	<MUL>	multiplied with -"-
13	8192	$0008.0000	⟨en-1⟩|⟨en⟩	DIV	2	<DIV>	division by -"-
...	...	...	...	...	..	...	...

Fig. 14b

Function:	*OpCodes of: (machine-code compilation of these mnemonics, here a Motorola-Example)*
Func.1	CMPI 127,⟨E⟩ ; JLE ⟨+2⟩ ; MOVI #127,⟨E⟩ ; CMPI -128,⟨E⟩ ; JGE ⟨+2⟩ ; MOVI #-128,⟨E⟩
Func.2	TST ⟨E⟩ ; JGE ⟨+3⟩ ; MOVI #-1,⟨E⟩ ; JMP ⟨+5⟩ ; JGT ⟨+3⟩ ; MOVI #0,⟨E⟩ ; JMP ⟨+2⟩ ; MOVI #+1,⟨E⟩
Func.3	MOVI #31,D0 ; BTST D0,⟨E⟩ ; JEQ ⟨+3⟩ ; DJMP D0,⟨−2⟩ ; ADDI #1,D0 ; MOVE D0,⟨E⟩
Func.4	FILD ⟨E⟩ ; FSQRT ; FIST ⟨E⟩
Func.5	MOVE ⟨E-1⟩,−(A7) ; MOVE ⟨E⟩,⟨E-1⟩ ; MOVE (A7)+,⟨E⟩

Fig. 14c

Valuation-Function-Table: *[VFT - Table of the valuation-functions]*

column:	*datatype*	*value-range*	*meaning:*
Valuation_Function_ID *(PK)*	signed short	±32767	identifier of the valuation-function (energyspecif. neg.)
Valuation_Function_Type	char(1)	'E'|'A'	'E' = energy-valuation, 'A' = valuation for reaching closeness to programming-aim, ... (maybe further)
Valuation_Function_Mode	boolean	0|1	0 = SQL-mode ; 1 = machine-code mode
Valuation_Function	varchar2(99	≤99 Bytes	valuation-function for energy- or aim-attainment
execution_counter	integer	$0\text{-}2^{32}\text{-}1$	number of uses of this valuation-function
used_Functions_BitCode	number(19)	$0..2^{64}\text{-}1$	BitCodes of all subfunctions
Function_ID_Chain	varchar2(99	≤99 Bytes	chain of subfunctions (one byte ≙ one Function_ID)
avg_Func_execution_time	integer	$0\text{-}2^{32}\text{-}1$	average by val.-func.-execution needed clock-cycles
boundary_value_counter	integer	$0\text{-}2^{32}\text{-}1$	counter incremented if the result is -128 or +127
low_value_counter	integer	$0\text{-}2^{32}\text{-}1$	counter incremented if the result inside ±16
Valuation_Function_value	signed byte	-128..127	valuability of the valuation-function = SAC.*Self-Valuation_Aim/Energy(Valuation_Function, Values)*

Fig. 15a *Initial Entries for Energy-Valuation and Aim-Closeness-Valuation:*

ID	Ty	M	Valuation_Function
-1	'E'	0	MAX[MIN[SGN(*EnergyReg'-EnergyReg°*) · SQRT(*EnergyReg'-EnergyReg°*) −32· Π CLT(i).*Register_changed_BitCode* &(! 2^*Energy_Register_ID*) } , +127], -128]
0	'A'	0	MAX[MIN[16· Π CLT(i).*Register_changed_BitCode* & ADT.*all_dest_Register_BitCode* } +16· Π CLT(i).*Register_source_BitCode* & ADT.*all_source_Register_BitCode* } +32·ADT.*aim_fulfilled_Flag_Function*(Aim_ID) −CLT(i).*Processor_Mode_changed* −¼·CLT(i).*cycles_of_execution* −(CLT(i).*active*|*inactive_ChkSum_corrupt*) −(CLT(i).*Exception_vect_changed* > 0) −(CLT(i).*Number_of_Exception* > 0) −½(CLT(i).*OpCode_length_or_jump* > 4 or ≤ 0), +127], -128]

ex#	used_F._BitCode	Function_ID_chain	ex.T	bdy#	low#	F.Val
0	$189.0040.983B	2,5; 2,15; 12; 4,22,3,11,35,40,1,32,12; 11 ; 39	0	0	0	0
0	$EE9.0001.3AAA	3,11,42,35,1,16,12; 3,12,41,35,1,16,12,10; 43,1,32,10, 3,7,11; 3,16,1,5,13,11; 3,3,11; 3,5,11 3,5,5,10; 3,8,5,10; 3,9,1,0,37,11;3,9,1,5,38,11;39	0	0	0	0

Fig. 15b

Status of the Artificial Consciousness: *[SAC - status-values of the AC-program (only 1 row)]*

column:	*datatype*	*value-range*	*meaning:*
Programm_StartDate	timestamp	datime	date and time of the start of the AC-program
actual_Processor_Mode	integer	$0\text{-}2^{32}\text{-}1$	flags above CCR | control-register-bits
actual_CPT_index	byte	1..255	CBT(max(i) = *actual_CPT_Nr*) = actual CPT
CxT_counter	short	1..65535	number of creatings of the dynamic CxT-tables
Aims_total	short	1..65535	number of total programming-aims
Aims_soluted	short	0..65535	number of solved programming-aims
actual_Aim_ID	short	0..65535	ID of the actual programming-aim
Aim_Valuation_Mode	boolean	0|1	mode of the programming-aim-specific valuation-function: 0 = SQL-mode ; 1 = machine-code mode
Aim_Valuation_FunctionID	signed short	0..32767	actual VFT.*Valuation_Function_ID* referring the closeness to the programming-aim
Aim_Self_Valuation_Func	varchar2 (400)	max.400 Chars.	PL/SQL-valuation-function referring the efficiency of the valuation-function
Energy_Valuation_Mode	boolean	0|1	mode of the energyspecific valuation-function 0 = SQL-mode ; 1 = machine-code mode
Energy_Valuation_Func_ID	signed short	-1..-32768	actual VFT.*Valuation_Function_ID* for energy-valuation
Energy_Self_Valuation_Func	varchar2 (400)	max.400 Chars.	PL/SQL-valuation-function referring the efficiency of the energyspecific valuation-function
max_Valuation_Function	signed short	0..32767	highest ID of all valuation-functions in the VFT.
min_Valuation_Function	signed short	-1..-32768	lowest ID of all valuation-functions in the VFT.

Fig. 16

Energy-Learn-Table: *[ELT - appraises energyspecific actions in dependence of the used initial-conditions]*

column:	*datatype*	*value-range*	*meaning:*
Energy_action *(PK)*	number	$0..2^{128}-1$	max.16 byte opcode-combination of the action which changed the energy-register.
IniConNr *(PK)*	signed byte	-31..30	number of the used initial condition
Energy_before	integer	$0..2^{32}-1$	energy-register before execution
Energy_after	integer	$0..2^{32}-1$	energy-register after execution
min_Operations_BitCode	number	$0..2^{128}-1$	bitcode of the probably used opcodes.
max_Operations_BitCode	number	$0..2^{128}-1$	bitcode of the possibly used opcodes.
Register_changed_BitCode	number	$1..2^{128}-1$	bitcode of the by action changed registers.
Register_source_BitCode	number	$1..2^{128}-1$	bitcode of the probable source-registers.
used_cycles_of_execution	short	1..65535	needed clock cycles for the energyspecific action
Energy_valuation	signed byte	-128..127	result of the actual *VFT.Energy_valuation_Function*
Valuation_Function_ID	signed short	-1..-32768	used energyspecific valuation-function

Fig. 17

Energy-Base-Table: *[EBT - evaluation of the energyspecific actions]*

column:	*datatype*	*value-range*	*meaning:*
Energy_action *(PK)*	number	$0..2^{128}-1$	max.16 byte opcode-combination of the action which changed the energy-register.
Execution_counter	byte	0..255	number of the ELT-entries until now.
FatalError_counter	byte	0..255	number of the occurred fatal errors: fatal errors correlate the columns 3-7 of the above learn-table, except *Divide-Error* or *Overflow-Exc.*
low_Error_counter	byte	0..255	number of *Divide-Error*s or *Overflow* -Exceptions
avg_Energy_after	integer	$0..2^{32}-1$	average energy-value after the action
all_Reg_dest_BitCode	number	$0..2^{128}-1$	Ξ ELT.Register_changed_Bitcode $\forall$ ELT(OpCode)
cut_Reg_dest_BitCode	number	$0..2^{128}-1$	ζ ELT.Register_changed_Bitcode $\forall$ ELT(OpCode)
all_Reg_source_BitCode	number	$0..2^{128}-1$	Ξ ELT.Register_source_Bitcode $\forall$ ELT(OpCode)
cut_Reg_source_BitCode	number	$0..2^{128}-1$	ζ ELT.Register_source_Bitcode $\forall$ ELT(OpCode)
max_Operation_BitCode	number	$0..2^{128}-1$	Ξ ELT.max_Operation_BitCode $\forall$ ELT(OpCode)
min_Operation_BitCode	number	$0..2^{128}-1$	ζ ELT.min_Operation_BitCode $\forall$ ELT(OpCode)
all_Operation_BitCode	number	$0..2^{128}-1$	Ξ ELT.min_Operation_BitCode $\forall$ ELT(OpCode)
cut_Operation_BitCode	number	$0..2^{128}-1$	ζ ELT.max_Operation_BitCode $\forall$ ELT(OpCode)
max_write_value	integer	$0..2^{32}-1$	maximum of all energy-values after energy-action
min_write_value	integer	$0..2^{32}-1$	minimum of all energy-values after energy-action
avg_write_value	integer	$0..2^{32}-1$	average of all energy-values after energy-action
max_write_gradient	integer	$0..2^{32}-1$	maximum gradient of the changes energy-register
min_write_gradient	integer	$0..2^{32}-1$	minimum gradient of the changes energy-register ##
avg_write_gradient	integer	$0..2^{32}-1$	average gradient of the changes energy-register
equal_value_probability	signed byte	-128..127	probability of equal result of energyspecific action
avg_Energy_gradient	signed int	$\pm 2^{31}$	average value-gradient of this energyspecific action
equal_Gradient_probability	signed byte	-128..127	probability: gradient is constant
avg_cycles_of_execution	short	1..65535	average needed clock cycles for this action
avg_Energy_valuation	signed byte	-128..127	result of the actual *VFT.Energy_valuation_Function*
Valuation_Function_ID	signed short	-1..-32768	ID of the used energy-valuation-function

Fig. 18

3.2 Flowchart of the AC-Program:

3.2.1 CxT(i) value assignments:

ORT & CRT(i) value assingments:

ORT.Register_ID_dest := $\log_2$(*Bit*(OLT.Register_changed_Mask), *of the regarded changing*)
ORT.Register_ID_source := Register_ID(C°), if ORT.calculation_code > 0, *otherwise* -1
ORT.value_before_change := value(Register_ID_dest), *before opcode-execution*
ORT.value_after_change := value(Register_ID_dest), *after opcode-execution*
ORT.gradient_if_signed := MAX[MIN[ORT.value_after_change - ORT.value_before_change, +127], -128]
ORT.gradient_if_unsigned := MAX[MIN[ORT.value_after_change - ORT.value_before_change, +127], -128]
ORT.Operation_BitCode := $1\cdot$(Flags'≠Flags°)&&∀(V'=V°)&&[NF&&(V_I° < 0) || ZF&&(V_I° = 0)] $+2\cdot[(V_I' = -V_I°)\&\&\forall^-(V'=V°)] + 4\cdot[(V_I' = \sim V_I°)\&\&\forall^-(V'=V°)] + 8\cdot[(V_I' = OLB)\&\&\forall^-(V'=V°)]$ $+16\cdot[(V_I' = V_I° + OLB)\&\&\forall^-(V'=V°)] + 32\cdot[(V_I' = V_I° - OLB)\&\&\forall^-(V'=V°)] + 64\cdot[(V_I' = V_I° \cdot OLB)\&\&\forall^-(V'=V°)]$ $+128\cdot[(V_I' = V_I° / OLB)\&\&\forall^-(V'=V°)] + 256\cdot[(V_I' = V_I° \,\%\, OLB)\&\&\forall^-(V'=V°)] + 512\cdot[(V_I' = V_I° \cdot 2\hat{}OLB)\&\&\forall^-(V'=V°)$ $+2^{10}\cdot[(V_I' = V_I° / 2\hat{}OLB)\&\&\forall^-(V'=V°)] + 2^{11}\cdot[(V_I' = V_I° \,|\, OLB)\&\&\forall^-(V'=V°)] + 2^{12}\cdot[(V_I' = V_I° \,\&\, OLB)\&\&\forall^-(V'=V°)]$ $+2^{13}\cdot$(Flags'≠Flags°)&&∀(V'=V°)&&[(ZF=1)&&($2\hat{}OLB \,|\, \sim V°$) || (ZF=0)&&!($2\hat{}OLB \,|\, V_I°$)] $+2^{14}\cdot$(Flags'≠Flags°)&&∀(V'=V°)&&[NF&&(V_I° < OLB) || ZF&&(V_I° = OLB)] + $2^{15}\cdot[(V_I' = C_I°)\&\&\forall^-(V'=V°)]$ $+2^{16}\cdot[(V_I' = V_I° + C_I°)\&\&\forall^-(V'=V°)] + 2^{17}\cdot[(V_I' = V_I° - C_I°)\&\&\forall^-(V'=V°)] + 2^{18}\cdot[(V_I' = V_I° \cdot C_I°)\&\&\forall^-(V'=V°)]$ $+2^{19}\cdot[(V_I' = V_I° / C_I°)\&\&\forall^-(V'=V°)] + 2^{20}\cdot[(V_I' = V_I° \,\%\, C°)\&\&\forall^-(V'=V°)] + 2^{21}\cdot[(V_I' = 2^{C°} \cdot V_I°)\&\&\forall^-(V'=V°)]$ $+2^{22}\cdot[(V_I' = V_I° / 2^{C°})\&\&\forall^-(V'=V°)] + 2^{23}\cdot[(V_I' = V_I° \,|\, C_I°)\&\&\forall^-(V'=V°)] + 2^{24}\cdot[(V_I' = V_I° \,\&\, C_I°)\&\&\forall^-(V'=V°)]$ $+2^{25}\cdot$(Flags'≠Flags°)&&∀(V'=V°)&&[(ZF=1)&&($2\hat{}C_I° \,|\, \sim V_I°$) || (ZF=0)&&!($2\hat{}C_I° \,|\, V_I°$)] $+2^{26}\cdot$(Flags'≠Flags°)&&∀(V'=V°)&&[NF&&(V_I° < C_I°) || ZF&&(V_I° = C_I°)] $+2^{27}\cdot$[(IP' ≤ IP°) || (IP' > IP° + 4)]&&(Flags' = Flags°)&&∀(V'=V°) $+2^{28}\cdot$[(IP' ≤ IP°) || (IP' > IP° + 4)]&&(Flags' = Flags°)&&∀(V'=V°)&&(NF&!VF || !NF&VF) + <u>...∀**Jcc**(CCR)</u> $+2^{40}\cdot$\{[(IP' ≤ IP°) || (IP' > IP° + 4)]&&(V_I' = V_I° - 1) || (V_I' = -1)\}&&(Flags' = Flags°)&&∀(V'=V°) $+2^{41}\cdot$[(IP' = IP° ± OLB)&&((SP) = IP°)&&(Flags' = Flags°)&&$\forall^-$(V'=V°) $+2^{42}\cdot$[(IP' = -4(SP))&&(Flags' = Flags°)&&∀(V'=V°) + $2^{43}\cdot[(V_I' \neq V_I°)\&\&(!$ other_Integer_Operation_BitCode)] $+2^{44}\cdot[(V_F' \neq V_F°)\&\&(!$ other_FloatingPoint_Operation_BitCode)] + $2^{45}\cdot[$(CCR-Flags' = 0)&&∀(V_F' = 0)]· $+2^{46}\cdot[(V_I' = C_F°)\&\&\forall^-(V'=V°)] + 2^{47}\cdot[(V_F' = C_I°)\&\&\forall^-(V'=V°)] + 2^{48}\cdot[(V_F' = V_F° + C_I°)\&\&\forall^-(V'=V°)]$ $+2^{49}\cdot[(V_F' = V_F° - C_I°)\&\&\forall^-(V'=V°)] + 2^{50}\cdot[(V_F' = V_F° \cdot C_I°)\&\&\forall^-(V'=V°)] + 2^{51}\cdot[(V_F' = V_F° / C_I°)\&\&\forall^-(V'=V°)]$ $+2^{52}\cdot$(Flags'≠Flags°)&&∀(V'=V°)&&[NF&&(V_F° < C_I°) || ZF&&(V_F° = C_I°)] $+2^{53}\cdot$[(V_F' = 1.0) || (V_F' = 0.0) || (V_F' = π) || (V_F' = e)]&&$\forall^-$(V'=V°) $+2^{54}\cdot[(V_F' = -V_F°)\&\&(V_F° < 0)\&\&\forall^-(V'=V°)] + 2^{55}\cdot[(V_F' = C_F°)\&\&\forall^-(V'=V°)]$ $+2^{56}\cdot[(V_F' = V_F° + C_F°)\&\&\forall^-(V'=V°)] + 2^{57}\cdot[(V_F' = V_F° - C_F°)\&\&\forall^-(V'=V°)]$ $+2^{58}\cdot[(V_F' = V_F° \cdot C_F°)\&\&\forall^-(V'=V°)] + 2^{59}\cdot[(V_F' = V_F° / V_F°)\&\&\forall^-(V'=V°)] + 2^{60}\cdot[(V_F' \cdot V_F' = V_F°)\&\&\forall^-(V'=V°)]$ $+2^{61}\cdot[V_F' = \sin(V_F°)]\&\&\forall^-(V'=V°) + 2^{62}\cdot[V_F' = \cos(V_F°)]\&\&\forall^-(V'=V°) + 2^{63}\cdot[(V_F' = \text{atan}(V_F°))\&\&\forall^-(V'=V°)]$ $+2^{64}\cdot[(V_F' = V_F° \cdot 2\hat{}V_{F-1}°)\&\&\forall^-(V'=V°)] + 2^{65}\cdot[(V_F' = V_{F-1} \cdot \log_2(V_F°)\&\&\forall^-(V'=V°)]$ $+2^{66}\cdot$(Flags'≠Flags°)&&∀(V'=V°)&&[NF&&(V_F° < C_F°) || ZF&&(V_F° = C_F°)] + $2^{67}\cdot[(V_I' = C_S°)\&\&\forall^-(V'=V°)]$ $+2^{68}\cdot[(V_S' = C_I°)\&\&\forall^-(V'=V°)] + \ldots$, where V'=value_after_change ($\neg$Flags), V°=value_before_change C°=value(Register_ID_source). Here has to be checked over all Register_ID_source(eq.kind). Though equal Register-ID's in the PK several bits can be set. [p.e. because 4 = 2 + 2 = 2*2 = SHL(2) = ...]

Fig. 19

OLT & CLT(i) value assingments:

OLT.Processor_Mode_Changed := $\mathcal{N}$\{ EFlags$_{\pi}$/SR$_{\mu}$ & ! Ξ 2^CCR_Flags } > 0 || ORT.value_after_change(Register_ID *of a special-register*)
OLT.aim_valuation := VFT.Aim_Valuation_Function(SAC.Aim_Valuation_FunctionID, ORT.xxxxx, Registers_changed_BitCode, Registers_source_BitCode, min_Operations_BitCode, max_Operations_BitCode, used_cycles_of_execution, ...)
CLT(n).gradient_aim_valuation := CLT(n).aim_valuation – CLT(n-1).aim_valuation
all other column-assignments are declared adequate in the OLT-description in fig.6.

Fig.20

OBT & CBT(i) value-assingments:

OBT.Execution_counter := Execution_counter+1
OBT.FatalError_counter := FatalError_counter +(0 < OLT.Number_of_Exception ≠ Divide_Error, Overflow) || OLT.active_ChkSum_corrupt || OLT.inactive_ChkSum_corrupt || OLT.Exception_vect_changed || OLT.Processor_Mode_changed)
OBT.Jump_longOp_probability := MAX[MIN[Jump_probability +(OLT.OpCode_length_or_jump ≤ 0) +(OLT.OpCode_length_or_jump>4), +127], -128]
OBT.avg_OpCode_jump_length := (execution_counter*avg_OpCode_jump_length +akt.OpCode_jump_length) / (execution_counter+1)
OBT.OpCode_len_unconfirmed := OpCode_len_unconfirmed || (avg_OpCode_length ≠ act.OpCode_length)
OBT.avg_cycles_of_execution := (execution_counter * avg_cycles_of_execution + act.cycles_of_execution) / (execution_counter+1)
OBT.exec_cycles_unconfirmed := exec_cycles_unconfirmed || (avg_cycles_of_execution ≠ act.cycles_of_execution)
OBT.Register_write_probability := MAX[MIN[Register_write_probability +2*[(min.Reg.ID ≤ ORT.Column_ID_OLT ≤ max.Reg.ID)&& ORT.value_before_change ≠ ORT.value_after_change] -1, +127], -128]
OBT.Register_copy_probability := MAX[MIN[MIN(Register_copy_probability +2*[(min.Reg.ID ≤ ORT.Column_ID_OLT ≤ max.Reg.ID)&& ORT.value_before_change ≠ ORT.value_after_change &&(min.Reg.ID ≤ ORT.Column_ID_source ≤ max.Reg.ID)] -1, +127], -128]
OBT.Memory_write_probability := MAX[MIN[Memory_write_probability +2*[(min.Adr.Reg.ID ≤ ORT.Column_ID_OLT ≤ max.Adr.Reg.ID)&& ORT.value_before_change ≠ ORT.value_after_change] -1, +127], -128]
OBT.Memory_copy_probability := MAX[MIN[Memory_copy_probability +2*[(min.Adr.Reg.ID ≤ ORT.Column_ID_OLT ≤ max.Adr.Reg.ID)&& ORT.value_before_change ≠ ORT.value_after_change &&(min.Adr.Reg.ID ≤ ORT.Column_ID_source ≤ max.Adr.Reg.ID)] -1, +127], -128]
OBT.Reg_to_Mem_probability := MAX[MIN[Reg_to_Mem_probability +2*[(min.Adr.Reg.ID ≤ ORT.Column_ID_OLT ≤ max.Adr.Reg.ID)&& ORT.value_before_change ≠ ORT.value_after-change &&(min.Reg.ID ≤ ORT.Column_ID_source ≤ max.Reg.ID)] -1, +127], -128]
OBT.Mem_to_Reg_probability := MAX[MIN[Mem_to_Reg_probability +2*[(min.Reg.ID ≤ ORT.Column_ID_OLT ≤ max.Reg.ID)&& ORT.value_before_change ≠ ORT.value_after_change &&(min.Adr.Reg.ID ≤ ORT.Column_ID_source ≤ max.Adr.Reg.ID)] -1, +127], -128]
OBT.Multi_Reg_write_prob := *like in* Register_write_probability , *but with min.2 appropriate* ORT.Column_ID_OLT *-entries.*
OBT.Multi_Mem_write_prob := *like in* Memory_write_probability , *but with min.2 appropriate* ORT.Column_ID_OLT *-entries.*
OBT.Multi_Reg_to_Mem_prob := *like in* Reg_to_Mem_probability , *but with min.2 appropriate* ORT.Column_ID_OLT+Column_ID_source *-entries.*
OBT.Multi_Mem_to_Reg_prob := *like in* Mem_to_Reg_probability , *but with min.2 appropriate* ORT.Column_ID_OLT+Column_ID_source *-entries.*
OBT.xxx_Reg_source|dest_BitCode: *see table-description*
OBT.xxx_calculation_BitCode: *see table-description*
OBT.max_write_value := MAX(max_write_value, ORT.value_after_change)
OBT.min_write_value := MIN(min_write_value, ORT.value_after_change)
OBT.avg_write_value := (execution_counter * avg_write_value + ORT.value_after_change) / (execution_counter+1)
OBT.max_write_gradient := MAX(max_write_gradient, ORT.value_after_change - ORT.value_before_change)
OBT.min_write_gradient := MIN(min_write_gradient, ORT.value_after_change - ORT.value_before_change)
OBT.avg_write_gradient := (execution_counter * avg_write_gradient + ORT.value_after_change - ORT.value_before_change) / (execution_counter+1)
OBT.evaluated_source_[Num]Register := *probability-function*(xxx_Reg_source_BitCode, confirmation_counter)

OBT.evaluated_dest_Register[2] : = *probability-function*(xxx_Reg_dest_BitCode, confirmation-counter)
OBT.evaluated_Operation_ID : = *probability-function*(xxx_Operation_BitCode, confirmation-counter)
OBT.Confirmation_counter : = Confirmation_counter + *exist*(*equivalent* OLT+ORT–*eintry with lower* IniConNr)
OBT.max_aim_valuation : = MAX(max_aim_valuation, OLT.aim_valuation)
OBT.avg_aim_valuation : = (execution_counter * avg_aim_valuation + OLT.aim_valuation) / (execution_counter + 1)
CBT(n).max_grad_aim_valuation : = MAX(CBT(n)max_aim_valuation, CLT(n).aim_valuation) – CBT(n-1).max_aim_valuation.
CBT(n).avg_grad_aim_valuation : = (execution_counter * CBT(n).avg_aim_valuation + CLT(n).aim_valuation) / (execution_counter + 1) – CBT(n-1).avg_grad_aim_valuation

Fig.21

3.2.2 ELT and EBT value-assignments:

ELT.max_Operations_BitCode : = OLT.max_Operations_OpCode
ELT.min_Operations_BitCode : = OLT.min_Operations_OpCode
ELT.Register_changed_BitCode : = OLT.Registers_changed_BitCode
ELT.Register_source_BitCode : = OLT.Registers_source_BitCode
ELT.Energy_Valuation : = VFT.Energy_valuation_Function(SAC.Energy_Valuation_FunctionID, Energy_after, Energy_before, Registers_changed_BitCode, Registers_source_BitCode, min_Operations_BitCode, max_Operations_BitCode, used_cycles_of_execution, ...)
ELT.Valuation_Function_ID : = *for calculation of* Energy_Valuation *used* VFT.Valuation_Function_ID
EBT.avg_Energy_after : = (execution_counter · avg_Energy_after + ELT.Energy_after) / (execution_counter + 1)
EBT.equal_value_probability : = equal_value_probability + 2·(avg_Energy_after = ELT.Energy_after) -1
EBT.avg_Energy_gradient : = (execution_counter · avg_Energy_gradient + ELT.Energy_after - ELT.Energy_before) / (execution_counter + 1)
EBT.equal_gradient_probability : = equal_gradient_probability + 2·(avg_Energy_gradient = ELT.Energy_after–ELT.Energy_before) -1
EBT.xxx_Operations|Registers_BitCode *see table-description*
EBT.avg_cycles_of_execution : = (execution_counter · avg_cycles_of_execution + ELT.used_cycles-_of_execution) / (execution_counter + 1)
EBT.avg_Energy_Valuation : = (execution_counter · avg_Energy_Valuation + ELT.Energy_valuation) / (execution_counter + 1)

Fig.22

3.2.3 Definitions needed to read the flowchart:

directives denotes a directive or a short sequence of directives.
⟨ *condition fulfilled ?* ⟩ *Yes:* branches horizontally, *No:* continue below.
(***continuing-label***) denotes a label to or from another part of the flowchart.
block of directives denotes a block of earlier defined directives.

In the flowchart because of the complexity not all things are described until the smallest detail, but the fundamental functionality is presented clear and comprehensible.
Self-evident things like closing a database-cursor or cofilling not explicit mentioned but existing table-columns (which do not need a special algorithm) are not performed additionally, because the meaning of these columns is already declared in 3.1.2 an their assignment-formulas in 3.2.1 or in 3.2.2.
In the flowchart means *"generate ORT-entry and actualise OBT"* what is already shown in the value-assignments in fig. 19-21.

Fig.23

3.2.4 AC-flowchart:

a.) Initial Preparations:

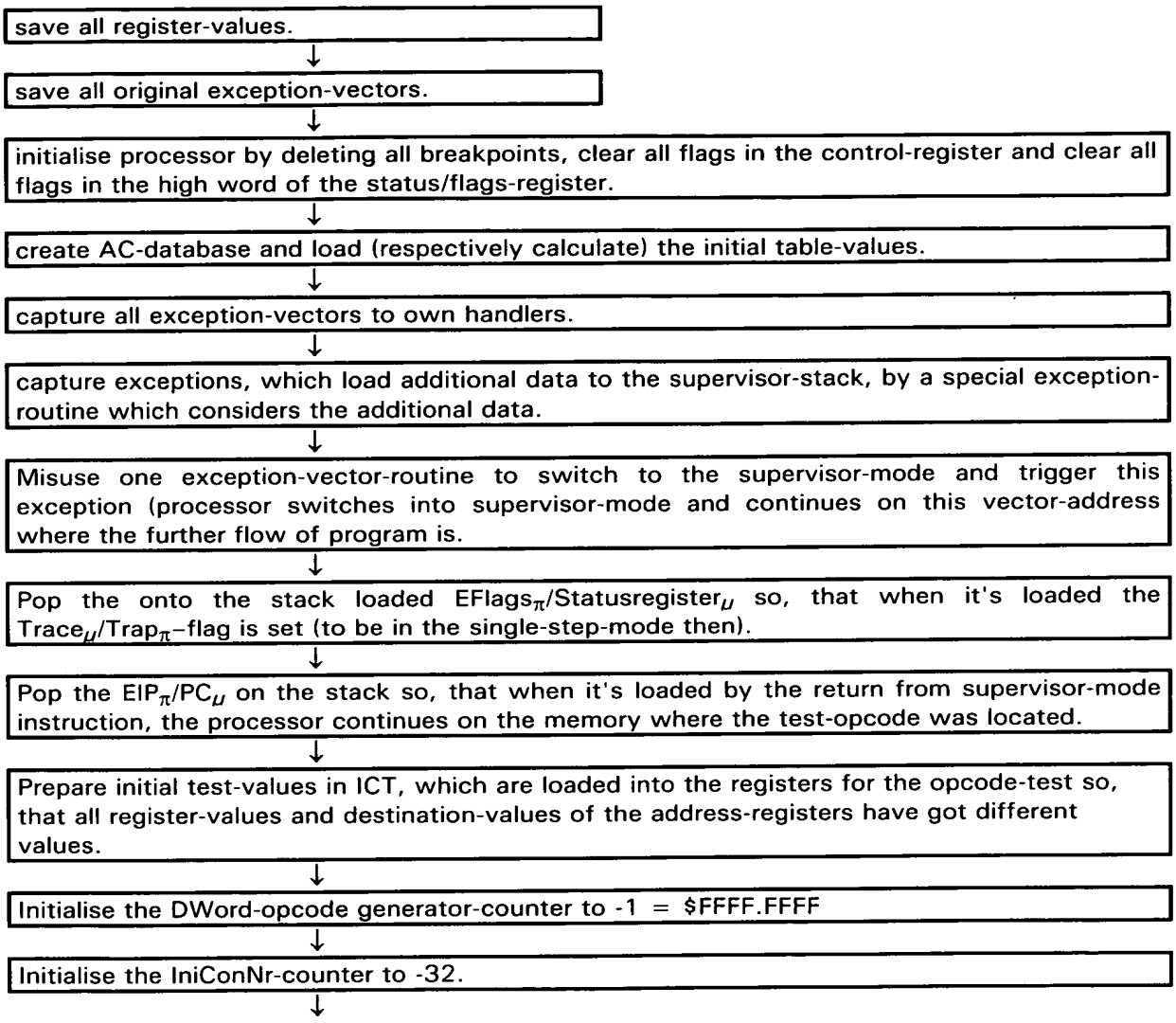

Fig.24a

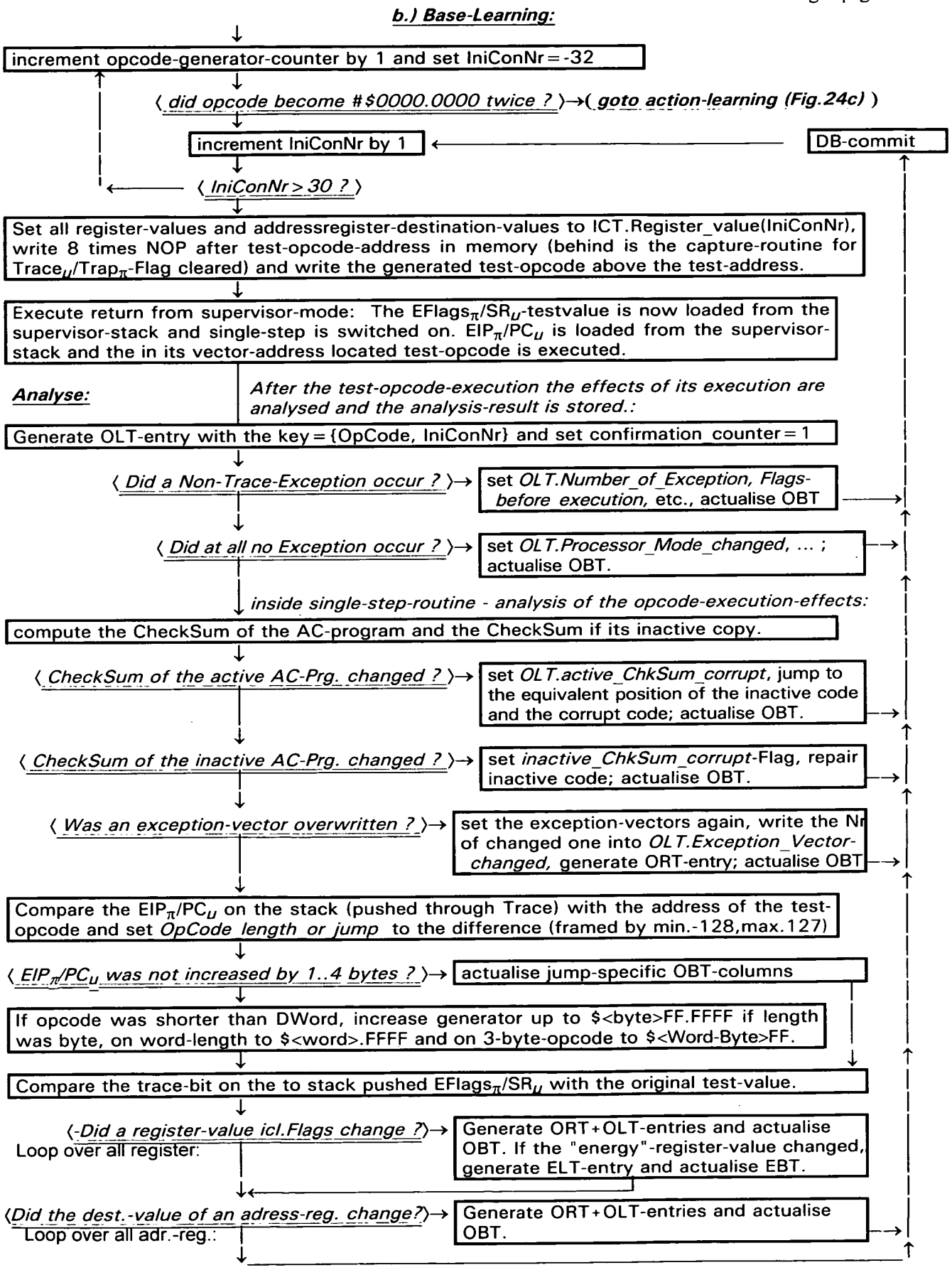
b.) Base-Learning:
increment opcode-generator-counter by 1 and set IniConNr = -32
⟨ did opcode become #$0000.0000 twice ? ⟩→(goto action-learning (Fig.24c))
increment IniConNr by 1
DB-commit
⟨ IniConNr > 30 ? ⟩
Set all register-values and addressregister-destination-values to ICT.Register_value(IniConNr), write 8 times NOP after test-opcode-address in memory (behind is the capture-routine for Trace$_U$/Trap$_\pi$-Flag cleared) and write the generated test-opcode above the test-address.
Execute return from supervisor-mode: The EFlags$_\pi$/SR$_U$-testvalue is now loaded from the supervisor-stack and single-step is switched on. EIP$_\pi$/PC$_U$ is loaded from the supervisor-stack and the in its vector-address located test-opcode is executed.
Analyse:
After the test-opcode-execution the effects of its execution are analysed and the analysis-result is stored.:
Generate OLT-entry with the key = {OpCode, IniConNr} and set confirmation_counter = 1
⟨ Did a Non-Trace-Exception occur ? ⟩→
set OLT.Number_of_Exception, Flags-before_execution, etc., actualise OBT
⟨ Did at all no Exception occur ? ⟩→
set OLT.Processor_Mode_changed, ... ; actualise OBT.
inside single-step-routine - analysis of the opcode-execution-effects:
compute the CheckSum of the AC-program and the CheckSum if its inactive copy.
⟨ CheckSum of the active AC-Prg. changed ? ⟩→
set OLT.active_ChkSum_corrupt, jump to the equivalent position of the inactive code and the corrupt code; actualise OBT.
⟨ CheckSum of the inactive AC-Prg. changed ? ⟩→
set inactive_ChkSum_corrupt-Flag, repair inactive code; actualise OBT.
⟨ Was an exception-vector overwritten ? ⟩→
set the exception-vectors again, write the Nr of changed one into OLT.Exception_Vector-changed, generate ORT-entry; actualise OBT
Compare the EIP$_\pi$/PC$_U$ on the stack (pushed through Trace) with the address of the test-opcode and set OpCode_length_or_jump to the difference (framed by min.-128,max.127)
⟨ EIP$_\pi$/PC$_U$ was not increased by 1..4 bytes ? ⟩→
actualise jump-specific OBT-columns
If opcode was shorter than DWord, increase generator up to $<byte>FF.FFFF if length was byte, on word-length to $<word>.FFFF and on 3-byte-opcode to $<Word-Byte>FF.
Compare the trace-bit on the to stack pushed EFlags$_\pi$/SR$_U$ with the original test-value.
⟨-Did a register-value icl.Flags change ?⟩→
Loop over all register:
Generate ORT+OLT-entries and actualise OBT. If the "energy"-register-value changed, generate ELT-entry and actualise EBT.
⟨Did the dest.-value of an adress-reg. change?⟩→
Loop over all adr.-reg.:
Generate ORT+OLT-entries and actualise OBT.

Fig.24b

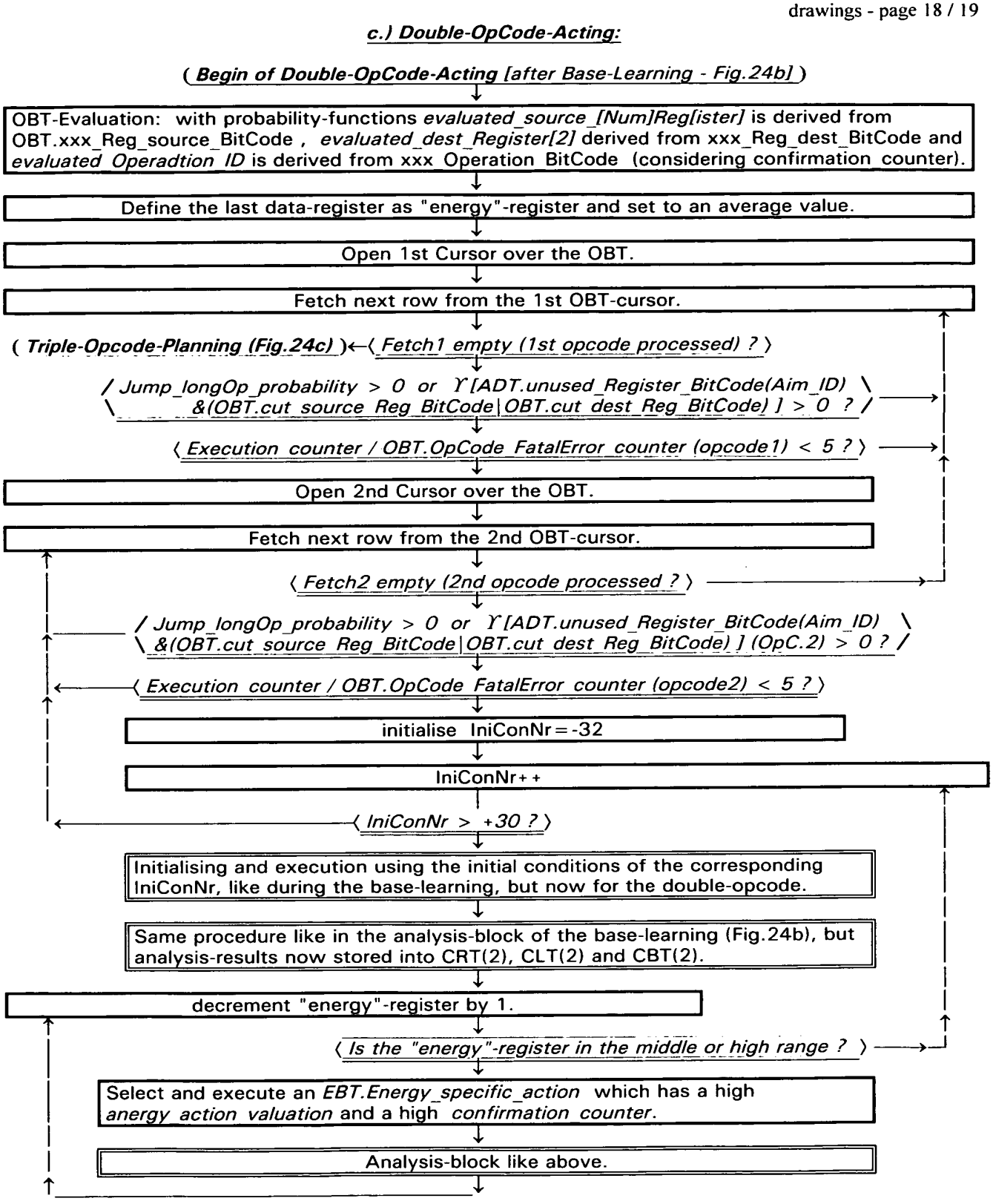
c.) Double-OpCode-Acting:
(Begin of Double-OpCode-Acting [after Base-Learning - Fig.24b])
OBT-Evaluation: with probability-functions evaluated_source_[Num]Reg[ister] is derived from OBT.xxx_Reg_source_BitCode , evaluated_dest_Register[2] derived from xxx_Reg_dest_BitCode and evaluated_Operadtion_ID is derived from xxx_Operation_BitCode (considering confirmation_counter).
Define the last data-register as "energy"-register and set to an average value.
Open 1st Cursor over the OBT.
Fetch next row from the 1st OBT-cursor.
(Triple-Opcode-Planning (Fig.24c))
⟨ Fetch1 empty (1st opcode processed) ? ⟩
/ Jump_longOp_probability > 0 or ϒ[ADT.unused_Register_BitCode(Aim_ID) \ &(OBT.cut_source_Reg_BitCode|OBT.cut_dest_Reg_BitCode)] > 0 ? /
⟨ Execution_counter / OBT.OpCode_FatalError_counter (opcode1) < 5 ? ⟩
Open 2nd Cursor over the OBT.
Fetch next row from the 2nd OBT-cursor.
⟨ Fetch2 empty (2nd opcode processed ? ⟩
/ Jump_longOp_probability > 0 or ϒ[ADT.unused_Register_BitCode(Aim_ID) \ &(OBT.cut_source_Reg_BitCode|OBT.cut_dest_Reg_BitCode)] (OpC.2) > 0 ? /
⟨ Execution_counter / OBT.OpCode_FatalError_counter (opcode2) < 5 ? ⟩
initialise IniConNr=-32
IniConNr++
⟨ IniConNr > +30 ? ⟩
Initialising and execution using the initial conditions of the corresponding IniConNr, like during the base-learning, but now for the double-opcode.
Same procedure like in the analysis-block of the base-learning (Fig.24b), but analysis-results now stored into CRT(2), CLT(2) and CBT(2).
decrement "energy"-register by 1.
⟨ Is the "energy"-register in the middle or high range ? ⟩
Select and execute an EBT.Energy_specific_action which has a high anergy_action_valuation and a high confirmation_counter.
Analysis-block like above.

Fig.24c

d.) Triple-OpCode-Planning:

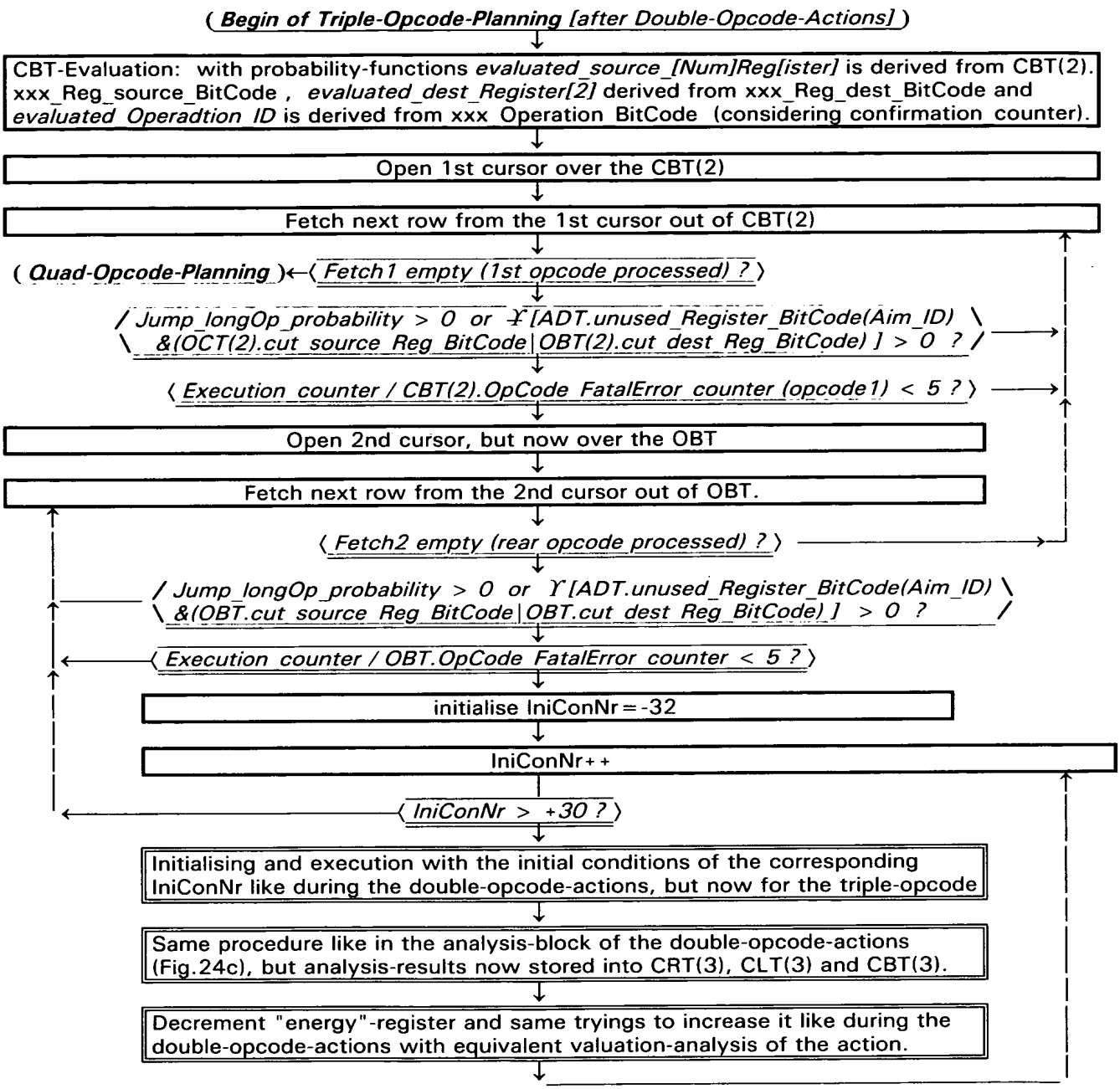

Fig.24d

Procedure for higher combinations analogous, using CxT(n), where n = sum_of_opcodes.